DeepPy: Pythonic deep learning

Larsen, Anders Boesen Lindbo

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

DeepPy: Pythonic deep learning

Anders Boesen Lindbo Larsen
Department of Applied Mathematics and Computer Science
Technical University of Denmark
abll@dtu.dk

DTU Compute Technical Report-2016-6, ISSN: 1601-2321

July 28, 2016

Abstract

This technical report introduces DeepPy – a deep learning framework built on top of NumPy with GPU acceleration. DeepPy bridges the gap between high-performance neural networks and the ease of development from Python/NumPy. Users with a background in scientific computing in Python will quickly be able to understand and change the DeepPy codebase as it is mainly implemented using high-level NumPy primitives. Moreover, DeepPy supports complex network architectures by letting the user compose mathematical expressions as directed graphs. The latest version is available at `http://github.com/andersbll/deeppy` under the MIT license.

1 Introduction

Under the term *deep learning*, the renaissance of neural networks has lead to impressive performance gains across a wide range of machine learning tasks. The comeback of neural networks is often accredited to practical matters summarized as 1) more data, 2) more compute power and 3) better software. Addressing the last point, Python-based libraries in particular have become popular for deep learning because the language is well-suited for scientific experimentation.

Current popular deep learning frameworks in Python offer fast numerical operations (on GPUs) thanks to run-time code generation and/or back-ends in high-performance languages, e.g. *Theano* [15] and *TensorFlow* [1]. This approach circumvents Python deficiencies such as slow interpretation and the global interpreter lock. However, there are also two notable drawbacks to this approach: First, the frameworks are not based on the de facto standard for numerical programming, *NumPy* [12]. Second, using the frameworks does not feel Pythonic since the back-end structures and operations cannot be accessed and changed with the flexibility that characterizes pure Python code.

In contrast to the approach above, DeepPy tries to provide a more Pythonic user experience by separating the high-performance primitives at the NumPy level such that only the individual array operations are performed by breaking out of Python. To leverage the computational power of GPUs, DeepPy is built on top of *CUDArray* [9] which implements a subset of the NumPy library. Other frameworks like *Chainer* [17], *Brainstorm* [5], and *Neon* [10] have a similar approach to DeepPy but blur the lines between numerical library and deep learning library which arguably leads to added complexity.

Among non-Python alternatives, the *Torch* framework and ecosystem [3] have the closest resemblance to DeepPy by allowing the user to build network functions as directed graphs.

2 Library overview

In this section, we briefly introduce the library design to give the bigger picture. Because changes to the library are likely to occur, we keep this section to the point and refer to the implementation on Github for more details.

Graph expression submodule, `deeppy.expr`

Central to deep learning frameworks are the methods for constructing network architectures. In the submodule `deeppy.expr`, DeepPy exposes methods and classes for the user to build mathematical expressions using operators and functions with semantics imitating NumPy. We recommend the user to think of the import statement `import deeppy.expr as ex` as being similar to `import`

`numpy as np`. Unlike NumPy, however, we are only building expressions at this point and postponing the actual computations until the entire model has been setup.

The mathematical expression that comprise a neural network is represented as a directed acyclic graph. After having built the network functions and defined a loss function for a model, we setup the expression graph to prepare for the computations. During the setup, all nodes in the expression graph infer their shape from their inputs in order to catch errors and to allow the nodes to setup their internal configuration. E.g. convolution operations may require benchmarking to determine the fastest computation kernels for the given input shapes. In order to perform shape inference, all inputs nodes in the graph must therefore know their array shape at this point (note that this is not required when building the graph). As the final task during setup, we sort the graph topologically yielding the forward and backward passes that allow us to perform automatic differentiation. That is, to calculate first-order derivatives of the model parameters wrt. the loss.

The submodule `deeppy.expr.nnet` contains operations typical for neural networks. These include activation and loss functions, layers with parameters (fully connected and convolution), batch normalization [6], and dropout [14].

Parameter class, `deeppy.Parameter`

Parameters are trainable variables which we learn by optimization. The parameter abstraction contains an array holding its values. It also exposes a method to obtain the latest gradient computed for the parameter. During training, we step the parameter values in the negative direction of the gradient according to minimize the loss function.

Feed class, `deeppy.Feed`

During training of a model, we must continuously transfer data to memory since datasets often are too large to reside in memory. The feed abstraction specifies an interface for serving input arrays to the model.

Model class, `deeppy.Model`

DeepPy's model abstraction is very basic as it mainly consists of a method for calculating parameter gradients given an input (e.g. a mini-batch from the training data). The intention behind the model abstraction is to wrap the expression graph that constitute the model in order to make it easy to use. E.g., for a more concrete model like `deeppy.models.FeedForwardNet`, we would add a `predict()` method to calculate the output for some given input.

Training submodule, `deeppy.train`

This submodule contains functionality for training models with popular gradient descent update rules like Adam [7] and RMSProp [16].

3 Speed

DeepPy relies on CUDArray to accelerate array operations. For performance critical operations like matrix multiplications and convolutions, CUDArray uses cuBLAS [11] and cuDNN [2] kernels meaning that DeepPy is on par with other deep learning frameworks. Because CUDA kernels run asynchronously from the CPU, we can hide the slowness of the Python code launching kernels. However, this also means that synchronous CUDA functions can hinder performance significantly and should be avoided.

4 Usage examples

We encourage the reader to study the examples that come bundled with DeepPy. These include feedforward, and recurrent networks, variational autoencoders [8,13], and generative adversarial networks [4].

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

[3] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Process-*

ing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014.

[5] K. Greff, R. K. Srivastava, and J. Schmidhuber. Brainstorm: Fast, Flexible and Fun Neural Networks, Version 0.5, 2015.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[8] D. P. Kingma and M. Welling. Stochastic gradient vb and the variational auto-encoder. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

[9] A. B. L. Larsen. CUDArray: CUDA-based NumPy. Technical Report DTU Compute 2014-21, Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2014.

[10] Nervana Systems. Neon, 2016.

[11] Nvidia. cuBLAS library v. 7.5, Sept. 2015.

[12] T. E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[13] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1278–1286, 2014.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[15] The Theano Development Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. Vlad Serban, D. Serdyuk, S. Shabanian, É. Simon, S. Spieckermann, S. Ramana Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang. Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, May 2016.

[16] T. Tieleman and G. Hinton. Lecture 6.5 – rmsprop: Divide the gradient by a running average of its recent magnitude. In *Neural Networks for Machine Learning*, 2012.

[17] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.